A MULTI-OBJECTIVE PERSPECTIVE ON JOINTLY TUNING HARDWARE AND HYPERPARAMETERS

David Salinas **dsalina@amazon.com**
Valerio Perrone vperrone@amazon.com
Olivier Cruchant cruchant@amazon.com
Cedric Archambeau cedrica@amazon.com

ABSTRACT

In addition to the best model architecture and hyperparameters, a full AutoML solution requires selecting appropriate hardware automatically. This can be framed as a multi-objective optimization problem: there is not a single best hardware configuration but a set of optimal ones achieving different trade-offs between cost and runtime. In practice, some choices may be overly costly or take days to train. To lift this burden, we adopt a multi-objective approach that selects and adapts the hardware configuration automatically alongside neural architectures and their hyperparameters. Our method builds on Hyperband and extends it in two ways. First, we replace the stopping rule used in Hyperband by a non-dominated sorting rule to preemptively stop unpromising configurations. Second, we leverage hyperparameter evaluations from related tasks via transfer learning by building a probabilistic estimate of the Pareto front that finds promising configurations more efficiently than random search. We show in extensive NAS and HPO experiments that both ingredients bring significant speed-ups and cost savings, with little to no impact on accuracy. In three benchmarks where hardware is selected in addition to hyperparameters, we obtain runtime and cost reductions of at least 5.8x and 8.8x, respectively. Furthermore, when applying our multi-objective method to the tuning of hyperparameters only, we obtain a 10% improvement in runtime while maintaining the same accuracy on two popular NAS benchmarks.

1 INTRODUCTION

The goal of neural architecture search (NAS) is to automatically find well-performing deep neural networks in domains such as computer vision and natural language processing. While initial efforts focused on discovering configurations with high accuracy, further work extended NAS to optimize for inference latency which is key to enable embedded devices or smartphones (Tan et al., 2018; Elsken et al., 2019). Methodologies that tune configurations to optimize other criteria than accuracy for a fixed hardware have been referred to as hardware-aware NAS (see Benmeziane et al., 2021, for a survey). Recently, Wu et al. (2019) and Li et al. (2021) proposed benchmarks that track latency in addition to accuracy for various hardware choices in order to compare hardware-aware approaches.

While much work focused on finding good architectures given a *fixed* hardware target, to the best of our knowledge no work considered tuning hardware together with architectures (or hyperparameters) in this context. NAS often starts *after* choosing a hardware configuration (e.g., the type and numbers of GPUs to use) which can have a massive impact on runtime and cost. Figure 1 shows the measured runtime when training Resnet on Cifar10 with all the machines available on AWS for two different batch-sizes. This reveals two key aspects of the hardware selection problem. First, there is not a single best instance but a set of optimal trade-offs. For instance, for the large batch-size (right panel) the optimal instances are p3.2x and all g4dn instances as they are on the Pareto front, which means that using any other instance will be both more expensive and slower to train Resnet for 250 epochs and a batch size of 1024. Second, the performance is non-stationary with respect to the hyperparameters. Indeed, changing the batch-size causes the hardware performance to drastically change (the axis are logarithmic). An even stronger effect would be expected across different tasks, calling for methods that automatically find the best hardware alongside hyperparameter configurations.

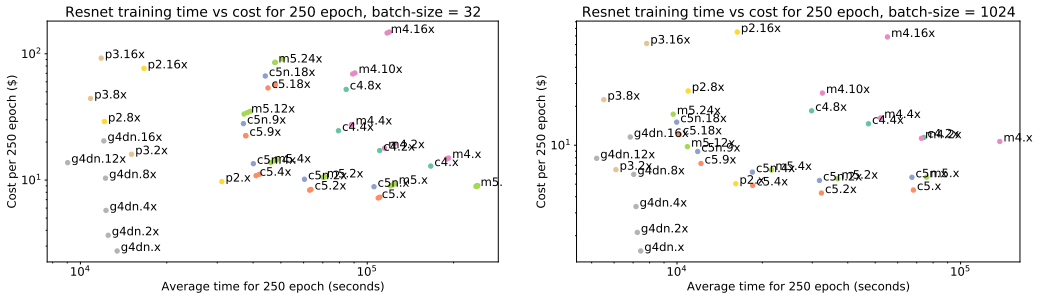

Figure 1: Time and cost to train Resnet on Cifar10 on AWS instance-types for a small (left) and a larger batch-size (right). The axis are in a logarithmic scale and colors indicate instance families. It can be observed that time and cost span several orders of magnitude.

This paper makes the following contributions:

- We extend Hyperband to handle multiple objectives by relying on non-dominated sorting, which is shown to be more efficient than previously-proposed scalarization approaches.
- We propose a new way to achieve transfer learning in the multi-objective setting by building a probabilistic approximation of the Pareto front.
- We show in experiments that tuning hardware alongside neural architectures significantly brings down the cost and runtime all else being equal.
- We also show that when only optimizing the hyperparameters, our multi-objective method is more efficient than standard multi-objective optimization algorithms.

2 Multi-objective Hyperband

Hyperband is a multi-fidelity hyperparameter optimization (HPO) algorithm that early stops unpromising configurations during training (Li et al., 2018). Random configurations are first evaluated at a small resource level (e.g., a small number of epochs). The fraction of configurations that is performing best is continued and then evaluated at a larger resource level (e.g., a larger number of epochs). This process is repeated until only a few are left and evaluated at the largest resource level. The pseudo-code of Hyperband is provided in the appendix. We present extensions to the sorting and the sampling strategy at the core of Hyperband, allowing us to tackle multi-objective problems.

Sorting configurations. Typically, sorting is done based on accuracy. However, with multiple objectives no strict ordering exists and one has to rely on multi-objective sort. A simple approach is scalarization: a mapping $s : \mathbb{R}^m \to \mathbb{R}$ is used to project the m objectives to a single value as proposed by Schmucker et al. (2020) and Cruz et al. (2020) for Hyperband. Instead, we propose to use non-dominated sorting (Emmerich & Deutz, 2018). To the best of our knowledge, we are the first to apply it to multi-objective Hyperband. Figure 2 illustrates this approach, which iteratively removes points from the Pareto front and breaks ties via a heuristic priority rule aiming at covering the Pareto front as efficiently as possible. In our case, we use a greedy epsilon-net strategy that first picks the point minimizing the first objective and then iteratively selects the farthest point from the current set. This approach comes with theoretical guarantees on coverage and sparsity (Clarkson, 2006). The pseudo-code of the method is given in the appendix.

Sampling configurations. Hyperband samples new configurations at random. More efficient strategies build a surrogate model of the optimization objective to direct the sampling (Falkner et al., 2018; Tiao et al., 2020) or use transfer learning to exploit evaluations from related tasks, such as when tuning the same algorithm on different datasets (Perrone et al., 2019). We propose a strategy related to the latter suitable for the multi-objective case. We assume a list of n hyperparameters $x_1, \ldots, x_n$ are evaluated on N tasks and we denote $y_i^j \in \mathbb{R}^m$ the m objectives recorded for $1 \leq i \leq n$ and $1 \leq j \leq N$. To sample new configurations, we proceed in three steps. First, we normalize

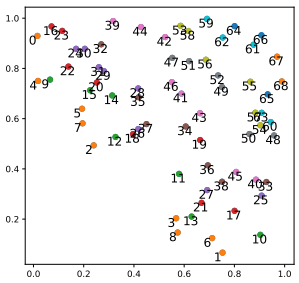

Figure 2: Non-dominated sorting. Labels denotes ranks of different points, colors denotes Pareto front groups.

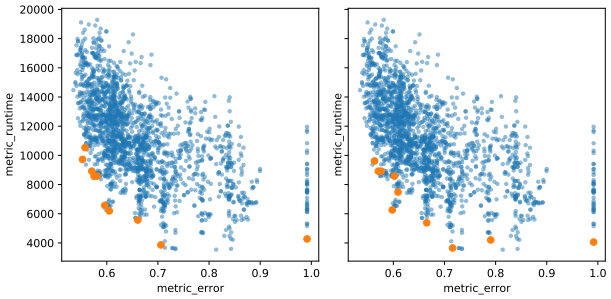

Figure 3: Points obtained when sampling surrogates estimates $\tilde{z}(x_1), \ldots, \tilde{z}(x_n)$ and picking the first configurations according to non-dominated sorting for two different seeds.

	fcnet-cloud			*nas201-cloud*			*hw-nas*		
	error	runtime	cost	error	runtime	cost	error	runtime	cost
HB	0.26 (100)	26.74 (100)	274.13 (100)	0.56 (100)	16.65 (100)	53.39 (100)	0.55 (100)	6.71 (100)	1.68 (100)
+Parego	0.34 (77)	3.93 (679)	55.54 (493)	0.65 (85)	2.35 (707)	19.21 (277)	0.66 (83)	1.04 (645)	0.48 (352)
+RW	0.33 (78)	4.85 (550)	54.77 (500)	0.61 (90)	3.42 (487)	21.36 (250)	0.64 (86)	1.38 (488)	0.53 (314)
+HV	0.31 (84)	9.21 (290)	80.54 (340)	0.58 (95)	11.16 (149)	30.61 (174)	0.61 (91)	3.25 (206)	0.86 (193)
+ND	0.26 (99)	21.72 (123)	153.97 (178)	0.56 (98)	13.06 (127)	38.75 (137)	0.56 (99)	5.38 (124)	1.22 (137)
+tr	0.28 (91)	4.57 (585)	4.84 (5666)	0.54 (102)	2.38 (700)	6.62 (806)	0.55 (100)	1.11 (605)	0.09 (1861)
+ND+tr	0.28 (91)	4.67 (572)	4.60 (5964)	0.56 (100)	1.99 (836)	6.52 (819)	0.56 (99)	0.91 (736)	0.08 (2115)

Table 1: Error, runtime in hours and cost in dollars when training Hyperband and multi-objective variants for hardware-aware benchmark. Percentage improvement over HB is shown in parenthesis.

observations with $z_i^j = \psi_j(y_i^j)$, where $\psi_j = \Phi^{-1} \circ F_j$ is the normalization used by Salinas et al. (2020) with Φ the Gaussian CDF and F_j the empirical CDF of $y_1^j, \ldots, y_n^j$. Second, we build a surrogate model of the conditional distribution $p(z|x_i) \approx \mathcal{N}(\tilde{\mu}_i, \tilde{\sigma}_i^2)$, where $\tilde{\mu}_i = \mathbb{E}_j[z_i^j]$ and $\tilde{\sigma}_i^2 = \mathbb{E}_j[z_i^j - \tilde{\mu}_i]^2$. While the procedure assumes the data is collected on a grid, it could be adapted through a surrogate model such as an MLP if the hyperparameters are not all evaluated on the different tasks (Salinas et al., 2020). Third, we sample the performance for all hyperparameters $\tilde{z}(x_1), \ldots, \tilde{z}(x_n)$ according to $p(z|x_i)$ and pick the top configurations according to the non-dominated sorting. Figure 3 illustrates the sampling of configurations.

3 Experiments

Benchmarks. We run experiments on five NAS benchmarks, the first two *nas201* (Dong & Yang, 2020) and *fcnet* (Klein & Hutter, 2019) contain neural architectures evaluated on multiple datasets for all possible fidelities (number of epochs). For those two datasets, we optimize for hyperparameters minimizing error and runtime. In addition, we use *hw-nas* (Li et al., 2021), which contains runtime for all 15625 neural architectures of *nas201* evaluated on 6 different edge devices. We also add cost per hardware which is minimized in addition to error and runtime. Finally, we build two new benchmarks *fcnet-cloud* and *nas201-cloud* that contain the estimated runtime and cost for all hyperparameters of *nas201* and *fcnet* on 37 hardware choices available on AWS (containing both cpu and gpu machines; we refer to the appendix for more details). For these three benchmarks, we optimize for error, runtime and cost, with the search space containing the hardware type in addition to hyperparameters. Note that cost and runtime cannot be merged as different hardware comes with different cost per hour.

Results. Table 1 and 2 report the average objectives obtained after running Hyperband (HB) and the following multi-objective variants for 30 seeds. We compare with different scalarizations, namely Parego (+Par) (Knowles, 2006), the random-weight approach (+RW) used for multi-objective multi-fidelity optimization by Schmucker et al. (2020), the scalarization of Golovin & Zhang (2020) which enjoys theoretical regret guarantees, and the non-dominated sort approach we propose (+ND). In addition, we report results of HB when using the transfer learning-based sampling described in the

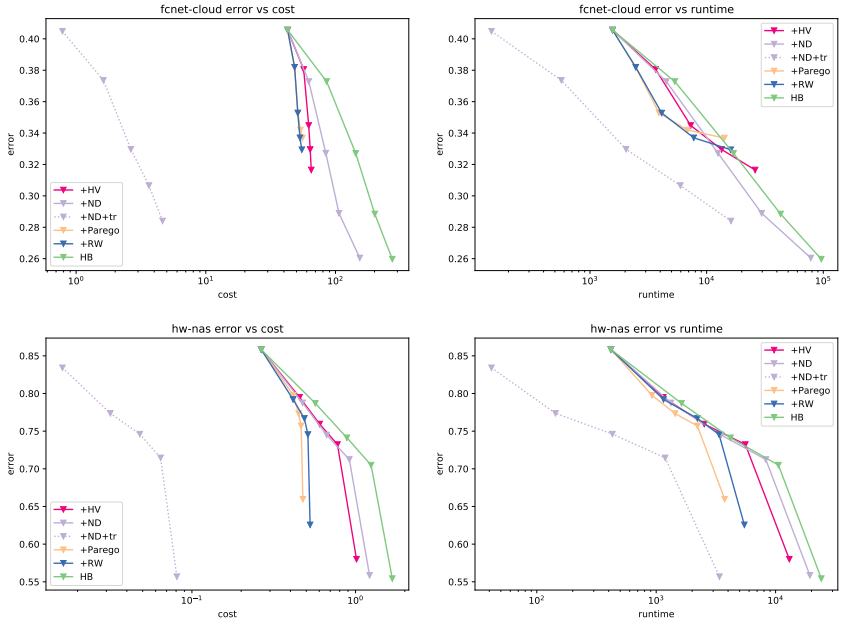

Figure 4: Error and cost (left, in dollars) or runtime (right, in seconds) when running HB with different strategies for *fcnet-cloud* (top) and *hw-nas* (bottom). Non-dominated sorting and transfer learning allow us to get similar final accuracy while significantly reducing cost and runtime, especially when we leverage observations from related tasks.

previous section (+tr) and when using in addition non-dominated sorting (+ND+tr) in contrast to ranking only with accuracy.

Table 2 shows that multi-objective non-dominated sorting reaches very similar errors as when minimizing only the accuracy, while also reducing the runtime. In the case where we only the hyperparameters, and thus not the hardware, we obtain runtime speed-ups of 20% and 10%, respectively, for *fcnet* and *nas201* while maintaining the same accuracy. In the benchmarks where we also tune the hardware (Table 1), the gains are all greater than 20% while we maintain a very similar accuracy. Other scalarization approaches methods save runtime/cost but a large decrease of accuracy is observed. We conjecture this is due to their independent ranking of configurations, which is outperformed by dominated sorting. Finally, the multi-objective transfer learning approach that we propose (+ND+tr) enables to save several order of magnitude in runtime and cost as can be seen in Figure 4 which compares all baselines in terms of average error against runtime/cost at all resource levels on *nas201-cloud* and *hw-nas*.

	nas201		*fcnet*	
	error	runtime	error	runtime
HB	0.26 (100)	14.86 (100)	0.56 (100)	5.39 (100)
+Parego	0.34 (77)	3.15 (471)	0.69 (80)	2.78 (193)
+RW	0.33 (77)	3.59 (413)	0.62 (90)	4.06 (132)
+HV	0.30 (86)	5.79 (244)	0.57 (97)	4.77 (112)
+ND	0.26 (99)	12.30 (120)	0.56 (99)	4.91 (109)
+tr	0.24 (109)	4.47 (332)	0.56 (100)	4.97 (108)
+ND+tr	0.24 (108)	3.91 (379)	0.56 (100)	4.76 (113)

Table 2: Methods comparison when tuning hyperparameters minimizing error and runtime. Percentage improvement over HB is shown in parenthesis.

4 Conclusion

We investigated the impact of hardware selection in the context of multi-fidelity hyperparameter optimization, showing that automatically selecting it brings large cost and runtime savings. The dramatic impact could be further pronounced when also considering distributed training or float precision. In addition, our method yields some improvements when searching only the hyperparameters. Future

work could include user constraints, such as not exceeding a cost budget, as well as model feasibility constraints, such as avoiding configurations prone to out of memory errors.

REFERENCES

Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search, 2021.

Kenneth L Clarkson. Nearest-neighbor searching and metric space dimensions. 2006.

André F. Cruz, Pedro Saleiro, Catarina Belém, Carlos Soares, and Pedro Bizarro. A bandit-based algorithm for fairness-aware hyperparameter optimization, 2020.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. 2020.

Thomas Elsken, Frank Hutter, and Jan Hendrik Metzen. Efficient multi-objective neural architecture search via Lamarckian evolution. *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–23, 2019.

Michael T.M. Emmerich and André H. Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing*, 17(3):585–609, 2018. ISSN 15729796. doi: 10.1007/s11047-018-9685-y.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. 2018. URL http://arxiv.org/abs/1807.01774.

Daniel Golovin and Qiuyi Zhang. Random Hypervolume Scalarizations for Provable Multi-Objective Black Box Optimization. 2020. URL http://arxiv.org/abs/2006.04655.

Aaron Klein and Frank Hutter. Tabular Benchmarks for Joint Architecture and Hyperparameter Optimization arXiv : 1905 . 04970v1 [cs . LG] 13 May 2019. (2018), 2019.

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006. doi: 10.1109/TEVC.2005.851274.

Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_0kaDkv3dVf.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:1–52, 2018.

Valerio Perrone, Huibin Shen, Matthias Seeger, Cédric Archambeau, and Rodolphe Jenatton. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. *arXiv*, (NeurIPS), 2019. ISSN 23318422.

David Salinas, Huibin Shen, and Valerio Perrone. A quantile-based approach for hyperparameter transfer learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8438–8448. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/salinas20a.html.

Robin Schmucker, Michele Donini, Valerio Perrone, Bilal Zafar Muhammad, and Cedric Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. *NeurIPS 2020 Workshop on Meta-learning*, 2020.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. 2018. URL http://arxiv.org/abs/1807.11626.

Louis Tiao, Aaron Klein, Thibaut Lienart, Cedric Archambeau, and Matthias Seeger. Model-based Asynchronous Hyperparameter and Neural Architecture Search. 2020. URL http://arxiv.org/abs/2003.10865.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNET: Hardware-aware efficient convnet design via differentiable neural architecture search. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:10726–10734, 2019. doi: 10.1109/CVPR.2019.01099.

A APPENDIX

For *nas201*, we use *ImageNet16-120* as test-task and use *Cifar10* and *Cifar100* to build a transfer-learning surrogate model. For *fcnet*, we use *protein_structure* as test-tasks and use the 3 remaining datasets to build the transfer-learning model. Those two datasets were chosen as they are the most difficult as the tasks are the most distinct from other available tasks, we will consider evaluating every combination in future work.

A.1 COST ESTIMATION OF *hw-nas*, *nas201-cloud* AND *fcnet-cloud*

***nas201-cloud* and *fcnet-cloud*.** As evaluations are only made given a single hardware choice for *nas201* and *fcnet*, we estimate the runtime that would have been obtained on different hardware. To this end, we measure the runtime per batch r_h of a similar model (resnet for *nas201*, MLP for *fcnet*) on 37 different instance types $h \in \mathcal{H}$ on AWS. The runtime of a configuration $x \in \mathcal{X}$ for a hardware $h \in \mathcal{H}$ is then estimated as $r(x, h) = r^{\text{original}}(x)\frac{r_h}{r_{h_{\text{ref}}}}$, where $r^{\text{original}}(x)$ is the original runtime measurement of the benchmark for the hyperparameter x and h_{ref} is a instance chosen to be close to the one used in the benchmark (`p2.xlarge` for *nas201* and `ml.c4.4xlarge` for *fcnet*).

In addition, we add a dollar cost objective $c(x, h) = \gamma_h r(x, h)$ based on the estimated runtime, where γ_h is the cost per second of the hardware h. In short, we are optimizing three objectives: error $e(x, h)$, runtime $r(x, h)$ and cost $c(x, h)$. The search space is $(x, h) \in \mathcal{X} = \mathbb{R}^d \times \mathcal{H}$, where $\mathcal{H}$ denotes the discrete set of possible instances and where the set of original hyperparameter is assumed in $\mathbb{R}^d$.

***hw-nas*.** We estimate the total cost for each hardware of (Li et al., 2021) as follow:

gpu1080	edgegpu	raspi4	edgetpu	pixel3	eyeriss	fpga
800$	499$	99.99$	129.99$	140$	2500$	2500$

The runtime of $r(x, h)$ of a hyperparameter x and a hardware h is estimated with

$$r(x, h) = r_{\textit{nas201}}(x)\frac{\text{lat}_{\textit{hw-nas}}(x, h)}{\text{lat}_{\textit{nas201}}(x)}$$

where $r_{\textit{nas201}}(x)$ denotes the runtime measured in *nas201* for the hyperparameter x, $\text{lat}_{\textit{hw-nas}}(x, h)$ the latency measured in (Li et al., 2021) and $\text{lat}_{\textit{nas201}}(x)$ denotes the latency measured in *nas201*.

Total costs of different hardware options were estimated by communication with the author of (Li et al., 2021) except for eyeriss whose price is unknown and estimated by us as the hardware is not on the market. To get the cost of a job $c(x, h)$ of an hyperparameter x and a hardware h, we multiply the runtime $r(x, h)$ by the total hardware cost divided by $3600 \times 24 \times 200$ (which assumes the hardware would have to be changed after 200 days).

A.2 PSEUDO-CODE

Hyperband. At its core, the multi-fidelity algorithm we propose builds on successive halving (SH), which we briefly describe. Let $f(x, r)$ be the target objective as a function of its hyperparameters x and the fidelity level r (e.g., the number of epochs at which $f(x)$ is evaluated). The overall procedure is described in Algorithm 1. First, all configurations are evaluated at the smallest resource level. Then, only the best $1/\eta$ configurations are kept in the set of configurations C (line 5). The fidelity parameter is then increased by a factor of η. This is repeated until the maximum budget is reached. SH is simple to implement and can be an efficient baseline, especially compared to standard HPO. However, selecting the number of number of configurations to evaluate is not easy in practice. For this reason, SH has been extended to Hyperband, which adds an outer iteration over different values of n and different budget, we give the pseudo-code from the method from (Li et al., 2018) on Alg. 1.

Importantly, the method only depends on sampling (line 5) and sorting configurations (line 10). The sorting can be extended with any scalarization method or with the non-dominated sort whose pseudo-code is given in Alg. 2.

Input: R, η (default to 3)
$s_{\text{max}} = \log_\eta(R)$, $B = (s_{\text{max}} + 1)R$;
for $s \in \{s_{max}, \dots, 0\}$ **do**
  $n = [\frac{B}{R} \frac{\eta^s}{s+1}], r = R\eta^{-1}$;
  $X =$ sample_configuration(n) ;
  for $i \in \{0, \dots, s\}$ **do**
    $n_i = [n\eta^{-i}]$;
    $r_i = r\eta^i$;
    $L = \{f(x, r_i) \mid \forall X \in X\}$;
    $X = \text{top}_k(X, L, [n_i/\eta])$;
  end
end

Algorithm 1: Pseudo-code for Hyperband.

Baseline details. We use the following scalarizations $s : \mathbb{R}^m \to \mathbb{R}$ as baseline to pick the top_k configurations:

- Linear (+RW) (Schmucker et al., 2020): $s(y) = \lambda^T y$ where λ is drawn randomly on a simplex when the function is called (we tried also searching for different values λ but the method perform poorly).
- Parego (+Par) (Knowles, 2006): $s(y) = \max_i \lambda_i y_i + \rho \sum_i \lambda_i y_i$ where $\rho > 0$ is a hyper-parameter set to a small value (0.05 in Knowles (2006) and our implementation) and λ is drawn randomly on a simplex when the function is called.
- Hypervolume (+HV) (Golovin & Zhang, 2020) $s(y) = \min_i(\max(0, y_i/\lambda_i))^m$ where λ is drawn randomly from the unit positive sphere when the function is called.

All objectives are centered with mean 0 and variance 1 with available observations. As Golovin & Zhang (2020) requires positive data, we also substract the minimum objective to map to positive values only for this scalarization.

The method we propose is non-parametric and both the non-dominated sort and the transfer learning approach have no hyperparameters (as the probabilistic surrogate of the Pareto front is built with empirical mean and variance). We use values in $[0, 29]$ for the seeds. The runtime of every method is bellow 1s for all benchmarks considered including running Hyperband for all iterations (excluding time to load tabular data).

Non-dominated sorting and epsilon-net pseudo-code. Alg. 2 and Alg. 3 give pseudo-code for non-dominated and epsilon-net sort, respectively.

Input: Set of objectives $y \in \mathbb{R}^{n \times m}$;
Output: Points sorted $\in \mathbb{R}^{n \times m}$;
if $n \leq 1$ **then**
  return y ;
else
  /* compute points of y that are in the Pareto front */
  $P =$ Pareto(y) ;
  /* use heuristic to break tie among points of P, call recursively on remaining points */
  return Sort(P) + ND-sort$(y \setminus P)$;
end

Algorithm 2: Pseudo-code for non-dominated sort.

Input: Set of objectives $y \in \mathbb{R}^{n \times m}$;
Output: Points sorted $\in \mathbb{R}^{n \times m}$;
if $n \leq 0$ **then**
  return y ;
else
  /* initialize with the point of y which has the lowest first coordinate */
  $i_1 = \text{argmin} y_{.,1}$;
  for $k \in [2, n]$ **do**
    /* pick point furthest away from the previously selected points */
    $i_k = \text{argmax}_{i \notin \{i_1, \ldots, i_{k-1}\}} d(y_i, \{y_{i_1}, \ldots, y_{i_{k-1}}\})$;
  end
  return $y_{i_1}, \ldots, y_{i_n}$;
end

Algorithm 3: Pseudo-code for Sorting in the non-dominated sort (epsilon-net).